NAG2-303

UCLA
School
of
Engineering
and
Applied
Science

ON-LINE RANGE PREDICTION SYSTEM (II)

September 1988

UCLA SCHOOL OF ENGINEERING AND APPLIED SCIENCE

(NASA-CR-185889) ON-LINE RANGE PREDICTION SYSTEM, PART 2 (California Univ.) 24 p CSCL 20E
N90-13771
Unclas
G3/36 0232368

ON-LINE RANGE PREDICTION SYSTEM*

(II)

* Research supported by NASA-Ames Dryden under Grant NAG 2-303, "<u>Survey and Analysis of X-Band Radar for Flight Research in the Western Aeronautical Test Range</u>", Principal Investigator: Nhan Levan, Research Assistant: Tayfun Cataltepe, Department of Electrical Engineering, 7732 Boelter Hall, UCLA.

SEPTEMBER 1988

Table of contents

I. INTRODUCTION:

This report is a follow-up to the interim technical report of July 1987 regarding the on-line range prediction system for the laser ranger at Crows Landing [1]. The on-line range prediction system is designed for providing a prediction of the target range in the case of a laser data dropout. It consists of real time implementation of a Kalman filter on an IBM PC/AT equipped with necessary hardware. The system was set up and tested at Crows Landing in the Fall of 1987.

This is a report of the improvements made on the on-line range prediction system during 1988. It is organized as follows. We begin by discussing and proposing solutions to the several problems encountered during system tests. Then, we explain the improvements made on the filter sotware, namely, accounting for the time lag and providing data continuously. Finally, we mention the ideas that can be considered in the future.

II. TEST RESULTS:

During the initial tests at Crows Landing, althoung the range output was stable, a jittering in the velocity output of the filter was observed. This problem was resolved during further tests performed at UCLA in Spring 1988 using an identical computer system and additional circuitry provided by Fred Shigemoto. The circuitry consisted of a 21 bit serial data generator simulating the incoming laser range data, and a serial to parallel converter that is also used in the original system at Crows Landing. Range and range velocity outputs of the system were displayed on a simple digital display circuit.

It was found that the jittering problem was caused by the "data ready" signal which was a square wave with period T, the sampling period of the filter. Thus, the "on" time of the ready pulse was also T seconds.But the sampling period is (and must be) longer than the time it takes to process a single data sample (2 milliseconds). Hence, when the system started checking for the ready pulse immediately after outputting the previously processed sample, it found a"high" signal even though no new sample had arrived. Consequently, the synchronization of the system with the arriving data samples was lost and this has resulted in unusal velocity estimates, or jittering.

For best performance, then, the ready signal should be a pulse having a width greater than 4 microseconds but less then the 2 milliseconds program execution time. Only in this case, each "data ready" pulse will correspond to a new incoming data sample and the synchronization of the system will be maintained.

III. PROGRAM TERMINATION:

The second problem is the termination of the program. At present, once it starts running it is not possible to stop the Kalman filter program using the keyboard commands except by system reset. Automatic stopping of the program occurs only when the range input is greater than 50 miles.

The assembly language routines and the main program are updated in a way that will enable normal termination of the execution upon request. The method is that an externally supplied termination request signal is recognized by the software and the program stops running. Bit 21, which is one of the unused bits of the 32 bit input data is employed for this purpose (see Fig.1).

A minimal external hardware consisting of a toggle switch between 0 and 5 volts is required. Bit 21 is already connected to bit 5 of port 2 on the input board (pin 43 on the input board connector). A high value set manually on this bit via the switch causes the input subroutine to return a special number to the main program which detects it, resulting in the program termination. The number sent is 2^{21} (counts), which corresponds to a range greater than 50 miles and since the filter software is already set to stop whenever the input range data is greater than 50 miles, this causes normal termination of the program.

Since the input subroutine is periodically called by the main program, a stop request is recognized by the upcoming subroutine call, hence it is always possible to stop the program. When program termination is requested through the switch, the data at the input port is ignored.

IV. TIME LAG BETWEEN INPUT AND OUTPUT:

There are two major improvements which have been made on the existing Kalman filter. The first one regards the output and its corresponding input. Since it takes a finite amount of time to process a sample (around 2 milliseconds), the range output corresponds to the input sample that has arrived 2 milliseconds earlier than the output instant (see Fig.2).

One way of accounting for this time lag is by propagating the range estimate using the velocity estimate. That is, at the instant $nT+\Delta$, $\hat{R}_{n+\Delta}$ is sent out to the port instead of $\hat{R}_n$,where

$$\hat{R}_{n+\Delta} = \hat{R}_n + \hat{V}_n \Delta \quad ,$$

and Δ is the amount of time lag, which is around 2 milliseconds execution time plus the time it takes to do the above calculation. $\hat{R}_n$ and $\hat{V}_n$ are the estimates of range and velocity, respectively, calculated by the Kalman filter using the input data that has arrived at the instant nT. The main program has been updated to do the propagation described. A listing can be found in the appendix.

V. CONTINUOUS PROPAGATION:

In its present form, after outputting the range and velocity estimates, the Kalman filter program waits for the next input sample. During this time period between the output and the arrival of the next sample, the system is idle in the sense that no data processing is performed. But, at the same time, the computer is not free to do any other task unless this task is integrated in the filter software.

This time period can be filled with range data updated by way of propagation. Currently, a multiplier and adder circuit is being designed at Moffett to perform this task. But, the propagation can be done in software rather than in hardware. The main program outputs the range and velocity estimates $\hat{R}_n$ (or $\hat{R}_{n+\Delta}$ as explained above) and $\hat{V}_n$ by passing them to an assembly language subroutine which configures a 32 bit data out of them and sends it to the output port. After this operation, the range $\hat{R}_n$ can be propagated using $\hat{V}_n$ <u>within the assembly language subroutine</u>, and then it can be sent to the port again. As in the previous case,

$$\hat{R}_{n+\tau} = \hat{R}_n + \hat{V}_n \tau$$

where τ is the time it takes to do the above calculation. This operation is repeated untill the next sample (or the data ready pulse) arrives as follows:

$$\hat{R}_{n+k\tau} = \hat{R}_{n+(k-1)\tau} + \hat{V}_n \tau \quad ,$$

where k is the repetition count (see Fig.3). Note that the velocity is assumed to be constant at $\hat{V}_n$ during this period of repeated propagation.

Statistically, the propagated range is the best possible estimate at that instant. Since the most recent observation (range input) is already utilized to obtain the Kalman filter estimates $\hat{R}_n$ and $\hat{V}_n$, and there is no new information available, the only way to estimate the range at any subsequent instant is by way of propagation [2].

The purpose of propagation is only to provide range data untill the next measurement arrives, and it does not affect the operation of the Kalman filter. When the new sample arrives, the filter uses the sample to update the estimate $\hat{R}_n$, not the propagated value. Also, since a linear time invariant model is assumed, the sampling interval T has to stay constant. If a sample arrives earlier than T seconds, the Kalman filter recognizes it as the next input and proceeds with the estimation, but this estimate is wrong since the calculations are based on constant sampling interval assumption.

The assembly language routines for handling input and output have been revised to do the tasks described above. In order to avoid too many subroutine calls and minimize the execution time, both of them are combined in a single assembly language subroutine.A listing is included in the appendix. The subroutine parameters are the output data to be configured, the range and velocity estimates, and the input data. The range and velocity estimates are passed onto the I/O subroutine in both real and 32 bit integer formats. Integer versions are used for configuring the output data. Real numbers are used in the propagation calculations. The IBM PC/AT is equipped with a 80287 math co-processor which is capable of handling high accuracy real number arithmetic with simple instructions in the assembly language level.

VI. CONCLUSION:

In this report, we proposed solutions to the problems encountered during the initial system tests, namely, jittering in the velocity output and the program termination. Also, we explained several ideas for improving the system performance. These are regarding the input-output time lag and more effective utilization of the system by providing more range estimates between the input samples.

As a result of these changes, the software had to be revised and a minimal external hardware had to be added. As before, either on-site tests or tests at at UCLA with the test circuitry are necessary to verify t~~o~~he functionality of the system.

In the future, more improvements can be made on the system. Incorporating an atmospheric refraction correction scheme has been on the agenda from the beginning. Once the proposed changes in this report are tested, a suitable refraction correction algorithm may be integrated into the Kalman filtering software. Also, since it is essentially a programmable real time data processing system, other applications of the system are possible.

VII. REFERENCES:

1. "On-line Range Prediction System", Interim Technical report for Survey and Analysis of X-Band Radar for Flight Research in the Western Aeronautical Test Range, NASA Ames Grant NAG 2-303, Principal Investigator: Nhan Levan, July 1987.
2. Balakrishnan, A. V., Kalman Filtering Theory, Optimization Sotware, Inc., Publications Division, New York, 1984.

VIII. FIGURES:

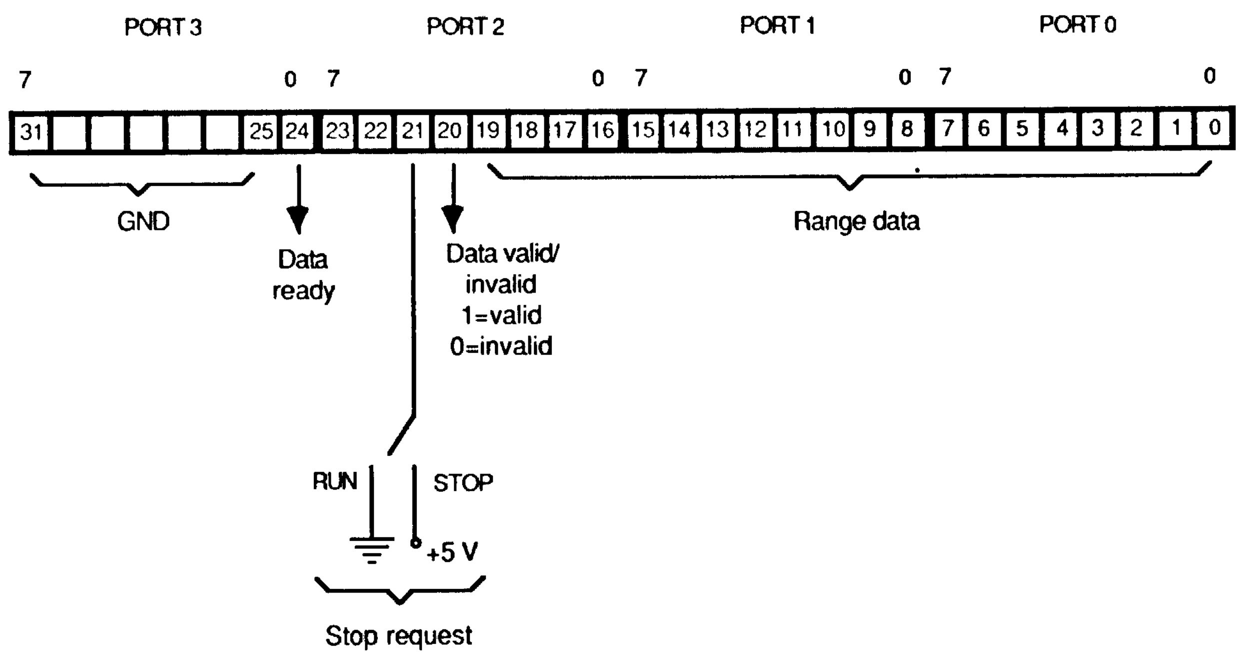

Fig. 1: Input data configuration

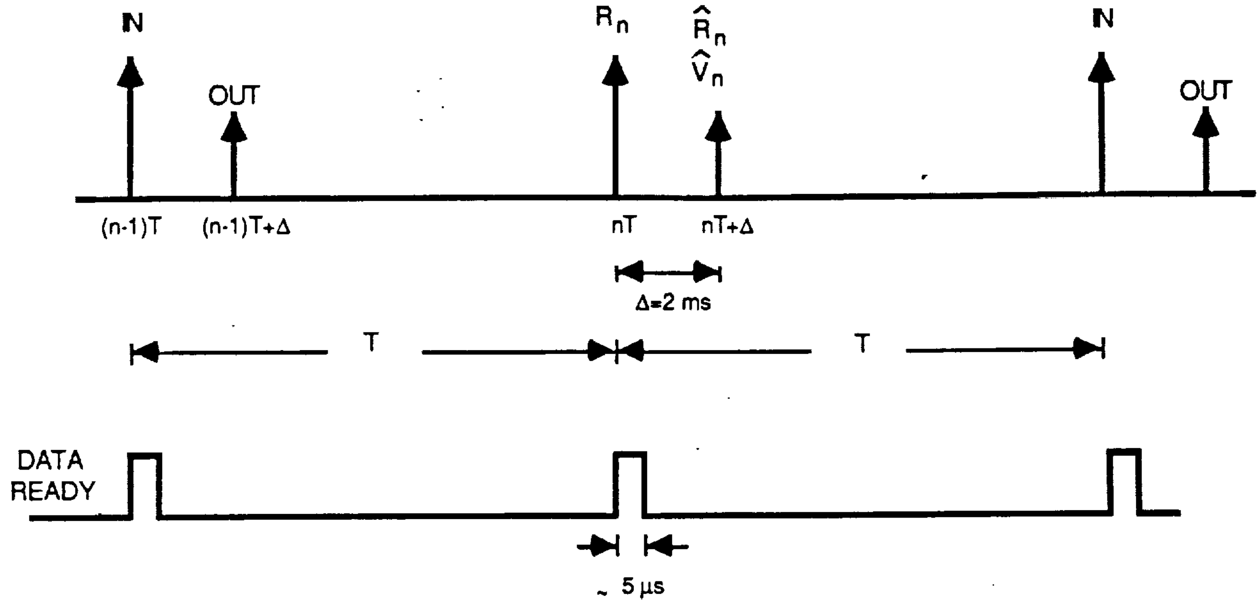

Fig. 2: Input -output time lag

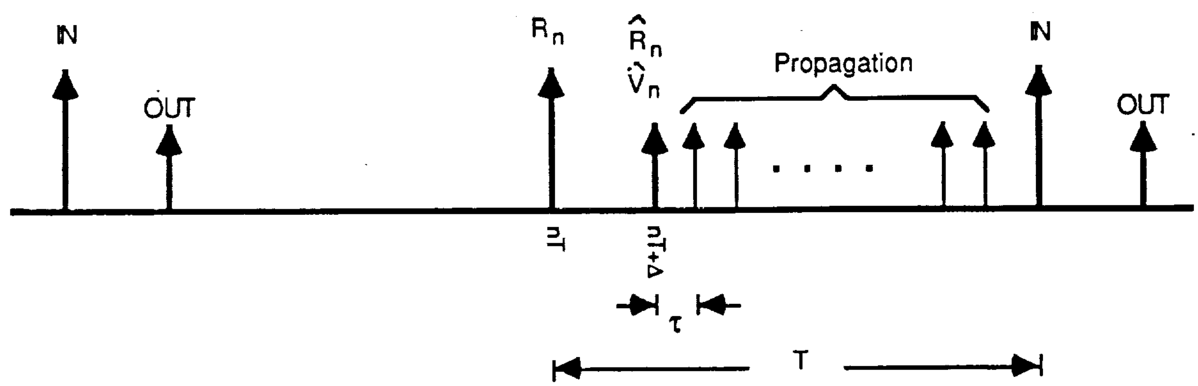

Fig. 3: Continuous propagation

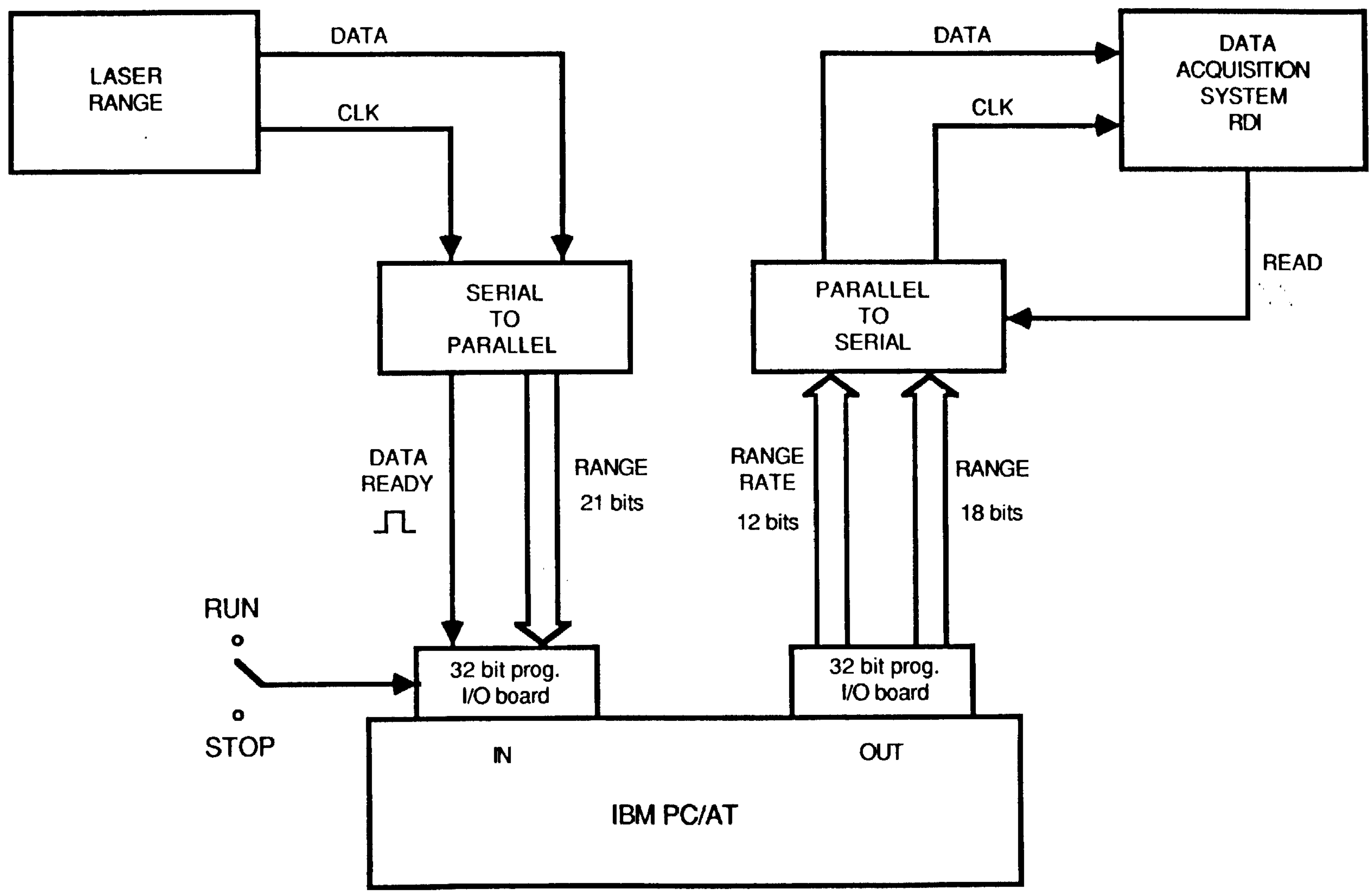

Fig. 4: System block diagram

IX. APPENDIX: PROGRAM LISTINGS

rangepr2.for

```
C*************************************************************
C                        VERSION II
C  THIS IS A TWO STATE KALMAN FILTER FOR LASER RANGE PREDICTION.
C  BY TAKING ADVANTAGE OF THE SYSTEM DYNAMICS, MATRIX CALCULATIONS
C  ARE ELIMINATED TO MINIMIZE EXECUTION TIME. SOME CONSTANTS THAT
C  ARE USED IN THE PROGRAM ARE INITIALIZED AT THE BEGINNING IN ORDER
C  TO SAVE CALCULATION TIME IN THE MAIN LOOP. ASSEMBLY LANGUAGE
C  ROUTINES ARE USED FOR INPUT/OUTPUT FOR HIGH SPEED. SOME PARAMETERS
C  ARE ENTERED INTERACTIVELY BY THE USER.
C
C  DEFINITIONS OF SYMBOLS
C     T            : SAMPLING INTERVAL
C     SCALE,OFFSET: PARAMETERS FOR CONVERTING COUNTS INTO FT
C     XPLS1, XPLS2: ELEMENTS OF STATE ESTIMATE VECTOR
C     XMNS1        : ONE STEP PREDICTION FOR RANGE
C     V,IV         : OBSERVED RANGE (FEET, COUNTS)
C     IV0          : OBSERVED RANGE AT THE PREVIOUS INSTANT (CNTS)
C     IXPLS1,
C     IXPLS2,KOUT : INTEGER VARIABLES USED FOR I/O ROUTINE CALLS
C     P11,P12,P22 : ELEMENTS OF ERROR COVARIANCE MATRIX
C     H11,H12,H22 : ELEMENTS OF PREDICTION ERROR COVARIANCE
C     FF11,
C     FF12,FF22   : ELEMENTS OF STATE NOISE COVARIANCE
C     GG          : OBSERVATION NOISE VARIANCE
C     SIGMA       : STRAIGHT LINE FLIGHT VARIANCE
C     MINVAR      : MINIMUM STATE NOISE VARIANCE
C     MAXVAR      : MAXIMUM STATE NOISE VARIANCE
C     Z1          : MAXIMUM INNOVATION AT WHICH MINVAR IS USED
C     Z2          : MINIMUM INNOVATION AT WHICH MAXVAR IS USED
C
C  ASSEMBLY LANGUAGE SUBROUTINES
C     INOUT       : READS THE 24 BIT DATA FROM THE INPUT PORT
C                   WHEN THE DATA READY SIGNAL COMES. WHEN THE READY
C                   SIGNAL IS NOT AVAILABLE, PROPAGATES THE RANGE
C                   (SECOND ARGUMENT) USING THE VELOCITY (THIRD ARG.).
C                   PROPAGATION LOOP TIME IS PLACED IN THE LAST
C                   ARGUMENT AND MUST NOT BE CHANGED.
C                   SENDS THE PROPAGATION TO OUTPUT PORT. READY
C                   SIGNAL IS CONNECTED TO BIT 24. THE INPUT DATA IS
C                   PLACED IN THE RETURN ARGUMENT. THE ARGUMENT MUST BE
C                   DECLARED AS 32 BIT INTEGER (SIXTH ARG). ALSO,
C                   CONFIGURES A 32 BIT OUPUT DATA (FIRST ARGUMENT)
C                   BY COMBINING THE 18 BITS OF THE FOURTH AND 12 BITS
C                   (MSB=SIGN) OF THE THIRD ARGUMENT. SENDS THE
C                   CONFIGURED ARGUMENT TO THE OUTPUT PORT. ALL INTEGER
C                   ARGUMENTS MUST BE DECLARED AS 32 BIT INTEGERS.
C                   THE SEVENTH ARGUMENT IS ONLY NEEDED IN THE ASSEMBLY
C                   ROUTINE AND NOTHING MUST BE PASSED TO SUBROUTINE
C                   IN THIS VARIABLE.
C     INDAT2      : READS THE FIRST TWO CONSEQUTIVE SAMPLES FOR
C                   INITIALIZATION DURING START UP, IF REQUESTED.
C*************************************************************
```

```
C
C  INITIALIZATIONS
      INTEGER*4 IV,IV0,IXPLS1,IXPLS2,KOUT,[illegible],IDUMMY
      REAL MINVAR,MAXVAR,MINVARA,MAXVARA
      TWOTAU=0.0002
1     PRINT*,'SCALE FACTOR                       = ?(FT/COUNT)  '
      READ*,SCALE
      IF (SCALE.LE.0.0) GO TO 1
      PRINT*,'OFFSET                             = ?(FT)        '
      READ*,OFFSET
2     PRINT*,'SAMPLING INTERVAL                  = ?(SEC)       '
      READ*,T
      IF (T.LT.0.003) THEN
         PRINT*,'SAMPLING INTERVAL MUST BE >= 3 MILLISEC'
         GO TO 2
      ELSE
      ENDIF
      PRINT*,'DO YOU WANT TO INITIALIZE THE RANGE AND RANGE VELOCITY'
      PRINT*,'OR USE THE FIRST TWO VALID SAMPLES FOR INITIALIZATION ?'
      PRINT*,' '
3     PRINT*,'0 = USE THE FIRST TWO SAMPLES'                        '
      PRINT*,'1 = INITIALIZE MANUALLY
      READ*,KTEST1
      IF (KTEST1.LT.0.OR.KTEST1.GT.1) GO TO 3
      IF (KTEST1.EQ.1) THEN
4        PRINT*,'INITIAL VALUE OF RANGE           = ?(FT)          '
         READ*,XPLS1
         IF (XPLS1.LE.0.0) GO TO 4
5        PRINT*,'INITIAL VALUE OF RANGE VELOCITY = ?(FT/SEC)   '
         READ*,XPLS2
         IF (ABS(XPLS2).GT.1000.0) GO TO 5
         IV=(XPLS1-OFFSET)/SCALE
         IV0=-1
      ELSE
      ENDIF
      PRINT*,' '
      PRINT*,'THE PARAMETERS ARE :'
      PRINT*,' '
      PRINT 100,SCALE
100   FORMAT (1X,'SCALE FACTOR      = ',F11.3,'  (FT/COUNT)')
      PRINT 101,OFFSET
101   FORMAT (1X,'OFFSET            = ',F11.3,'  (FT)')
      PRINT 102,T
102   FORMAT (1X,'SAMPLING INTERVAL= ',F11.3,'  (SEC)')
      GG=20.0
      SIGMA=31.68
      MINVAR=0.005*SIGMA
      MAXVAR=2.0*MINVAR
      Z1=0.01*SQRT(SIGMA)
      Z2=100.0*Z1
      P11=10.0
      P12=0.0
      P22=10.0
      PRINT*,' '
      PRINT*,' '
```

```
      PRINT*,'**********************************************************'
      PRINT*,' '
      PRINT*,' '
      PRINT*,' '
      PRINT*,'THE NOISE PARAMETERS ARE PRESET AS :'
      PRINT*,' '
      PRINT 103,GG
103   FORMAT (1X,'OBSERVATION NOISE VARIANCE (GG)                    = ',
     *F11.3)
      PRINT 104,MINVAR
104   FORMAT (1X,'MINIMUM STATE NOISE VARIANCE (MINVAR)              = ',
     *F11.3)
      PRINT 105,MAXVAR
105   FORMAT (1X,'MAXIMUM STATE NOISE VARIANCE (MAXVAR)              = ',
     *F11.3)
      PRINT 106,Z1
106   FORMAT (1X,'MAXIMUM INNOVATION AT WHICH MINVAR IS USED (Z1) = ',
     *F11.3)
      PRINT 107,Z2
107   FORMAT (1X,'MINIMUM INNOVATION AT WHICH MAXVAR IS USED (Z2) = ',
     *F11.3)
      PRINT 108,P11,P12
108   FORMAT (1X,'INITIAL VALUE OF ERROR COVARIANCE MATRIX (P)     = ',
     *F9.3,2X,F9.3)
      PRINT 109,P12,P22
109   FORMAT (1X,'                                                      ',
     *F9.3,2X,F9.3)
      PRINT*,' '
      PRINT*,'DO YOU WANT TO CHANGE THE NOISE PARAMETERS ? '
6     PRINT*,'0 = NO, 1 = YES                '
      READ*,KTEST
      IF (KTEST.LT.0.OR.KTEST.GT.1) GO TO 6
      IF (KTEST.EQ.0) GO TO 7
      PRINT 110
110   FORMAT(1X,'--ENTER ANY NEGATIVE VALUE TO LEAVE A PARAMETER',
     *' UNCHANGED--')
      PRINT*,' '
      PRINT*,'OBSERVATION NOISE VARIANCE (GG)                  = ? '
      READ*,GGA
      IF (GGA.LT.0.0) GO TO 9
      GG=GGA
9     PRINT*,'MINIMUM STATE NOISE VARIANCE (MINVAR)            = ? '
      READ*,MINVARA
      IF (MINVARA.LT.0.0) GO TO 10
      MINVAR=MINVARA
10    PRINT*,'MAXIMUM STATE NOISE VARIANCE (MAXVAR)            = ? '
      READ*,MAXVARA
      IF (MAXVARA.LT.0.0) GO TO 11
      MAXVAR=MAXVARA
11    IF (MAXVAR.LE.MINVAR) THEN
         PRINT*,'MINVAR MUST BE < MAXVAR, ENTER MINVAR AND MAXVAR AGAIN'
         PRINT*,' '
         GO TO 9
      ELSE
      ENDIF
```

```
      PRINT*,'MAXIMUM INNOVATION AT WHICH MINVAR IS USED (Z1) = ? '
      READ*,Z1A
      IF (Z1A.LT.0.0) GO TO 12
      Z1=Z1A
12    PRINT*,'MINIMUM INNOVATION AT WHICH MAXVAR IS USED (Z2) = ? '
      READ*,Z2A
      IF (Z2A.LT.0.0) GO TO 13
      Z2=Z2A
13    IF (Z2.LE.Z1) THEN
         PRINT*,'Z2 MUST BE > Z1, ENTER Z1 AND Z2 AGAIN'
         PRINT*,' '
         GO TO 11
      ELSE
      ENDIF
      PRINT*,'INITIAL VALUE OF ERROR COVARIANCE MATRIX :'
      PRINT*,'P(1,1)= ?      '
      READ*,P11A
      IF (P11A.LT.0.0) GO TO 14
      P11=P11A
14    PRINT*,'P(1,2)= ?      '
      READ*,P12A
      IF (P12A.LT.0.0) GO TO 15
      P12=P12A
15    PRINT*,'P(2,2)= ?      '
      READ*,P22A
      IF (P22A.LT.0.0) GO TO 16
      P22=P22A
16    IF ((P11*P22-P12**2).LE.0.0) THEN
         PRINT*,'ERROR COVARIANCE NOT POSITIVE DEFINITE, ENTER P AGAIN'
         PRINT*,' '
         GO TO 13
      ELSE
      ENDIF
7     TSQ=T**2
      TSQ2=TSQ/2.0
      TCUB=T**3/3.0
      TWOT=T*2.0
      Q0=MINVAR/TCUB
      Q1=MAXVAR/TCUB
      SLOPE=(Q1-Q0)/(Z2-Z1)
      ZINT=Q0-Z1*SLOPE
      IF (KTEST1.EQ.1) GO TO 30
17    CALL INDAT2(IV0)
      IF (IV0.LT.0) GO TO 17
      CALL INDAT2(IV)
      IF (IV.LT.0) GO TO 17
      IF (ABS(IV-IV0).GT.20) GO TO 17
      XPLS1=IV*SCALE+OFFSET
      XPLS2=(IV-IV0)*SCALE/T
      IV0=IV
      GO TO 30
```

```
C  MAIN LOOP STARTS
C
C  UPDATE STATE NOISE COVARIANCE (VARY LINEARLY WITH THE INNOVATION Z)
50     Z=ABS(XMNS1-V)
       IF (Z.GE.Z2) GO TO 20
       IF (Z.LE.Z1) GO TO 30
       QR=Z*SLOPE+ZINT
       GO TO 40
20     QR=Q1
       GO TO 40
30     QR=Q0
40     FF11=QR*TCUB
       FF12=QR*TSQ2
       FF22=QR*T
C
C  CALCULATE PREDICTION ERROR COVARIANCE
       H11=P11+TWOT*P12+P22*TSQ+FF11
       H12=P12+T*P22+FF12
       H22=P22+FF22
C
C  STATE PROPAGATION (PREDICTION)
       XMNS1=XPLS1+T*XPLS2
       E=H11+GG
       D1=GG/E
       D3=-H12/E
C
C  CONVERT COUNTS INTO FEET, CHECK FOR RUN/STOP,
C  CHECK FOR DATA VALID/INVALID, CHECK ONE STEP DIFFERENCE
       V=IV*SCALE+OFFSET
       IF (V.GT.260000.0) GO TO 99
       IF (IV.LT.0) GO TO 55
       IF (IV0.GE.0.AND.ABS(IV-IV0).GT.20) GO TO 55
C
C  STATE UPDATE (ESTIMATE)
       XPLS1=D1*XMNS1+H11*V/E
       XPLS2=D3*(XMNS1-V)+XPLS2
C
C  CALCULATE ERROR COVARIANCE
       P11=D1*H11
       P12=D1*H12
       P22=D3*H12+H22
       GO TO 56
C  IF OBSERVATION INVALID, USE ONE STEP PREDICTION
55     XPLS1=XMNS1
       P11=H11
       P12=H12
       P22=H22
C
C  SEND THE RANGE AND VELOCITY ESTIMATES TO THE OUTPUT PORT
C  READ THE NEW SAMPLE IF DATA IS READY
C  IF NOT READY, PROPAGATE THE RANGE ESTIMATE USING VELOCITY EST.
C  TAU IS THE TIME IT TAKES TO PROPAGATE (MULT. BY 2 SINCE VEL IS
C     DIVIDED BY 2)
56     IV0=IV
       PLS1=(XPLS1+XPLS2*.002-OFFSET)/SCALE
       IXPLS1=PLS1
       PLS2=XPLS2/(2.0*SCALE)
       IXPLS2=PLS2
       CALL INOUT(KOUT,PLS1,PLS2,IXPLS1,IXPLS2,IV,IDUMMY,TWOTAU)
C
C  GO BACK FOR THE NEXT DATA
       GO TO 50
99     STOP
       END
```

inout.asm

```
        PAGE ,132
        TITLE FORTRAN SUBROUTINE
        .8087
FRAMES  STRUC
DOUT    DD    ?                         ;DATA TO BE SENT TO THE PORT
RRANGE  DD    ?                         ;LOCATION OF RANGE (SHORT REAL)
RVEL    DD    ?                         ;LOCATION OF VELOCITY (SHORT REAL)
RANGE   DD    ?                         ;RANGE (32 BIT INTEGER)
VEL     DD    ?                         ;VELOCITY (32 BIT INTEGER)
KV      DD    ?                         ;INPUT DATA
Y       DD    ?                         ;VELOCITY IN SIGN/MAG. 16 BIT INT. FORM
TOTAU   DD    ?                         ;2*PROPAGATION LOOP TIME
FRAMES  ENDS
STACK   SEGMENT WORD STACK 'STACK'
        DB    64 DUP('MYSTACK')
STACK   ENDS
MYPRO   SEGMENT 'CODE'
        ASSUME CS:MYPRO,SS:STACK
INOUT   PROC FAR
        PUBLIC INOUT
        PUSH AX                         ;SAVE THE REGISTERS
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH SP
; CONFIGURE INPUT AND OUTPUT PORTS
        MOV  DX,230H            ;CONTROL REG OF OUTPUT PORT
        MOV  AL,0FH
        OUT  DX,AL              ;CONFIGURE OUTPUT PORTS
        MOV  DX,228H            ;CONTROL REG OF INPUT PORT
        SUB  AX,AX
        OUT  DX,AL              ;CONFIGURE INPUT PORTS
; STORE VELOCITY IN SIGN MAGNITUDE FORM FOR LATER USE
        LDS  SI,ES:VEL[BX]      ;LOAD ADDRESS OF VELOCITY
        MOV  AL,[SI]+3          ;GET SIGNED BYTE OF VEL
        CMP  AL,00H
        JNS  CONFIG             ;IF VEL +VE, GO ON WITH DATA CONFIGURATION
        MOV  AX,[SI]            ;ELSE, CONVERT TO SIGN (BIT 15)-MAG (0-14)
        NOT  AX
        ADD  AX,1
        OR   AX,8000H
        LDS  SI,ES:Y[BX]        ;ADDRESS OF SECOND VELOCITY LOCATION
        MOV  [SI],AX            ;STORE IN FIRST TWO BYTES OF VEL
        JMP  MANIP
 CONFIG: MOV AL,[SI]            ;FOR +VE VEL, PUT SIGN IN BIT 15
        LDS  SI,ES:Y[BX]        ;IN THE SECOND VELOCITY LOCATION
        MOV  [SI],AL
        LDS  SI,ES:VEL[BX]
        MOV  AL,[SI]+1
        AND  AL,7FH
        LDS  SI,ES:Y[BX]
        MOV  [SI]+1,AL
```

```
; CONFIGURE THE 32 BIT OUTPUT (DOUT)
MANIP:  LDS  SI,ES:RANGE[BX]  ;LOAD ADDRESS OF RANGE
        MOV  AX,[SI]          ;GET THE FIRST TWO BYTES OF RANGE
        LDS  SI,ES:DOUT[BX]   ;LOAD ADDRESS OF DATA TO BE SENT OUT
        MOV  [SI],AX          ;STORE THE FIRST TWO BYTES OF RANGE IN DOUT
        LDS  SI,ES:RANGE[BX]  ;RANGE ADDRESS AGAIN
        MOV  AL,[SI]+2        ;GET THE THIRD BYTE OF RANGE
        MOV  CL,2             ;ROTATION COUNT
        ROR  AL,CL
        LDS  SI,ES:Y[BX]      ;LOAD THE ADDRESS OF VELOCITY
        MOV  AH,[SI]          ;GET THE FIRST BYTE OF VELOCITY
        ROL  AX,CL            ;PUT THE CONFIGURED BYTE IN AH
        LDS  SI,ES:DOUT[BX]   ;ADDRESS OF DOUT
        MOV  [SI]+2,AH        ;STORE THE THIRD BYTE TO GO OUT
        LDS  SI,ES:Y[BX]      ;ADDRESS OF VELOCITY
        MOV  AX,[SI]          ;GET VELOCITY IN AX
        MOV  CL,3
        SAR  AX,CL            ;CONFIGURE FOURTH BYTE
        MOV  CL,4
        SAR  AH,CL
        MOV  CL,3
        SAR  AX,CL
        AND  AL,3FH
        LDS  SI,ES:DOUT[BX]
        MOV  [SI]+3,AL        ;STORE THE FOURTH BYTE TO GO OUT
; SEND THE DATA TO THE OUTPUT PORTS
        MOV  DX,231H          ;POINT TO PORT 0
        MOV  AL,[SI]          ;GET THE FIRST BYTE IN AL
        OUT  DX,AL            ;OUT FIRST BYTE TO PORT 0
        INC  DX               ;POINT TO PORT 1
        MOV  AL,[SI]+1        ;GET THE SECOND BYTE IN AL
        OUT  DX,AL            ;OUT SECOND BYTE TO PORT 1
        MOV  AL,[SI]+2        ;THIRD BYTE IN AL
        INC  DX               ;POINT TO PORT 2
        OUT  DX,AL            ;OUT THIRD BYTE TO PORT 2
        INC  DX               ;POINT TO PORT 3
        MOV  AL,[SI]+3        ;LAST BYTE IN AL
        OUT  DX,AL            ;LAST BYTE TO PORT 3
; READ THE DATA READY PULSE
        MOV  DX,22CH          ;POINT TO INPUT PORT 3
        IN   AL,DX
        RCR  AL,1             ;GET THE READY PULSE IN CARRY
        JC   READY            ;IF DATA READY, GO READ IT
; IF DATA NOT READY, PROPAGATE THE RANGE
        FINIT                 ;INITIALIZE THE CO-PROCESSOR
        LDS  SI,ES:RVEL[BX]
        FLD  DWORD PTR [SI]   ;VELOCITY TO TOP OF STACK
        LDS  SI,ES:TOTAU[BX]
        FMUL DWORD PTR [SI]   ;MULTIPLY BY TIME
        LDS  SI,ES:RRANGE[BX]
        FADD DWORD PTR [SI]   ;ADD TO THE RANGE
        FST  DWORD PTR [SI]   ;STORE THE REAL VALUE
        LDS  SI,ES:RANGE[BX]
        FISTP DWORD PTR [SI]  ;POP AND STORE THE 16 BIT INTEGER VERSION
        FWAIT
        JMP  MANIP            ;CHECK DATA READY PULSE AGAIN
```

```
; IF DATA READY, READ IT FROM THE INPUT PORTS
READY:  LDS  SI,ES:KV[BX]      ;LOCATION OF THE INPUT
        MOV  DX,22BH           ; POINT TO PORT 2
        IN   AL,DX
        AND  AL,3FH
        CMP  AL,20H            ;IS THERE A STOP REQUEST?
        JNS  TERM              ;IF STOP REQUESTED, SET UP FOR TERMINATION
        AND  AL,1FH
        CMP  AL,10H            ;IS THE DATA VALID?
        JS   PRED              ;IF NOT, GO TO PREDICTOR SET-UP
        AND  AL,0FH            ;IF VALID, MASK THE VALID/INVALID BIT
        MOV  [SI]+2,AL
        DEC  DX                ;READ THE REST OF THE INPUT
        IN   AL,DX
        MOV  [SI]+1,AL
        DEC  DX
        IN   AL,DX
        MOV  [SI],AL
        MOV  AL,00H
        MOV  [SI]+3,AL
        JMP  FIN
; INPUT DATA CONFIGURATION FOR INVALID SAMPLES
PRED:   MOV  AL,0FFH
        MOV  [SI]+3,AL
        JMP  FIN
; INPUT DATA CONFIGURATION FOR STOP REQUESTS
TERM:   SUB  AX,AX
        MOV  [SI],AX
        MOV  AX,0020H
        MOV  [SI]+2,AX
FIN:    POP  SP                ;RESTORE THE REGISTERS
        POP  DX
        POP  CX
        POP  BX
        POP  AX
        RET
INOUT   ENDP
MYPRO   ENDS
        END
```

inports2.asm

```
         PAGE ,132
         TITLE FORTRAN SUBROUTINE
FRAME    STRUC
NV       DD ?
FRAME    ENDS
STACK    SEGMENT WORD STACK 'STACK'
         DB   64 DUP('MYSTACK')
STACK    ENDS
MYPROG   SEGMENT 'CODE'
         ASSUME CS:MYPROG,SS:STACK
INDAT    PROC FAR
         PUBLIC INDAT
         PUSH AX              ;SAVE THE REGISTERS
         PUSH BX
         PUSH CX
         PUSH DX
         PUSH SP
         MOV  DX,228H         ;POINT TO CONTROL REG
         SUB  AX,AX           ;0 TO AX
         OUT  DX,AL           ;CONFIGURE PORTS FOR INPUT
         ADD  DX,4H           ;POINT TO PORT 3
REDY:    IN   AL,DX           ;READ THE DATA READY BIT
         RCR  AL,1            ;GET THE READY BIT IN CF FOR CHECKING
         JNC  REDY            ;IF DATA NOT READY, CHECK AGAIN
         LDS  SI,ES:NV[BX] ;LOCATION OF THE RANGE
         DEC  DX              ;POINT TO PORT 2
         IN   AL,DX           ;GET THE MSB OF THE DATA,BITS 16-23
         AND  AL,1FH          ;MASK THE UNUSED BITS (23,22,21)
         CMP  AL,10H          ;IS THE DATA VALID? (AL-10)
         JS   PRD             ;IF NOT VALID, GO TO PREDICTOR SET UP
         AND  AL,0FH          ;IF VALID, MASK THE VALID/INVALID BIT
         MOV [SI]+2,AL        ;STORE THE THIRD BYTE
         DEC  DX              ;POINT TO PORT 1
         IN   AL,DX           ;GET THE SECOND BYTE
         MOV  [SI]+1,AL       ;STORE THE SECOND BYTE
         DEC  DX              ;POINT TO PORT 0
         IN   AL,DX           ;GET THE FIRST BYTE
         MOV  [SI],AL         ;STORE THE FIRST BYTE
         MOV  AL,00H
         MOV  [SI]+3,AL       ;COMPLETE TO 32 BIT +VE INTEGER
         JMP  FINAL           ;GO TO RETURN
PRD:     MOV  AL,0FFH
         MOV  [SI]+3,AL       ;MAKE KV A NEGATIVE INTEGER
FINAL:   POP  SP              ;RESTORE THE REGISTERS
         POP  DX
         POP  CX
         POP  BX
         POP  AX
         RET
INDAT    ENDP
MYPROG   ENDS
         END
```